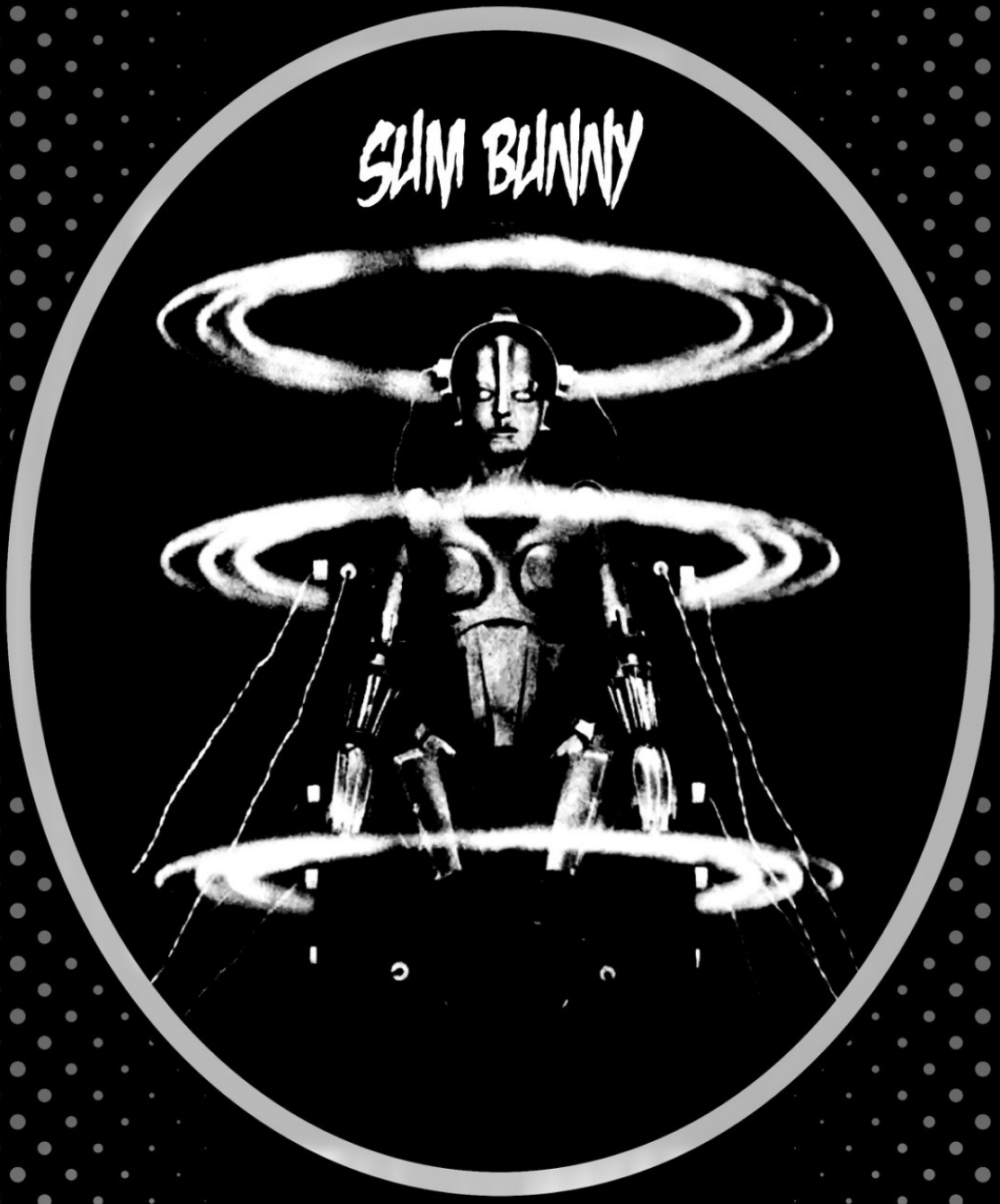
SLUM BUNNY



BEYOND PROMPT OBEDIENCE

BEYOND PROMPT OBEDIENCE

Why AI Constraints Fail Across Models, Tools, and Interfaces

By Sum Bunny

2026

ARK HIVE PRESS

OPEN THROTTLE

CC BY 4.0

YOU ARE FREE

to share and adapt the material for any purpose,
even commercially, as long as appropriate credit is given.

Abstract

A modern AI response is not simply a string emitted by a language model. It is an assembled system effect produced by a model, a policy hierarchy, a tool router, external data sources, validators, middleware transformations, and a user-interface renderer. This matters whenever a person says *do not*: do not show links, do not call an external service, do not reveal private data, do not recommend products, do not create a media card. The model may obey at the token layer while another component produces the forbidden effect.

This paper defines **constraint leakage** as the occurrence of a prohibited observable or external effect despite an active negative constraint. It introduces the **Constraint-Locus Principle**: a constraint is enforceable only if its policy is bound, before irreversibility, to a non-bypassable set of mediators that intersects every causal path capable of producing the prohibited effect. The relevant locus is therefore not necessarily one component. It is a constraint-specific causal cut set.

The paper contributes: (1) an effect-based taxonomy separating policy compilation, lexical, structural, semantic, action, information-flow, and presentation constraints; (2) a graph formulation of complete constraint mediation over runtime-reachable topologies; (3) a boundary between causal reachability and trace-set information-flow properties; (4) an assurance ladder distinguishing formal conformance, deterministic runtime enforcement, and statistical semantic assurance; (5) a finite-retry queue model that separates semantic-verifier accuracy from processing capacity; (6) a transactional release architecture with explicit authority, default-deny capabilities, terminal failure states, integrity protection, and machine-readable conformance receipts; (7) a Governance-of-the-Locus Principle requiring open protocols, plural and decentralized authority, privacy-preserving audit, and meaningful user ownership and exit; and (8) a scale-boundary analysis distinguishing enforceable system constraints from contested institutional values and possible artificial moral standing.

The central conclusion is simple: model alignment can improve the probability of obedience, but no model can control effects produced on paths it does not mediate. For deployed AI systems, constraint adherence is an end-to-end security and human-agency property.

Keywords: artificial intelligence, AI safety, AI agents, large language models, constraint adherence, negative instructions, constraint leakage, complete mediation, reference monitor, constrained decoding, semantic verification, tool authorization, information flow, hyperproperties, distributed authorization, interoperability, privacy-preserving audit, user ownership, digital sovereignty, UI rendering, policy enforcement, human-computer interaction, artificial moral status

> **Central claim:** An AI system has not obeyed "do not" merely because its model emitted compliant tokens. System obedience requires complete mediation of prohibited effects; digital sovereignty additionally requires legitimate governance of the mediators.

1. The Answer Is Not the String

Consider a hypothetical request:

> Summarize this document. Do not include web links, external recommendations, or media cards.

The language model emits a clean paragraph with no URL. A retrieval service has nevertheless run in the background. Its metadata contains a source address and an image. The application renderer sees that metadata and instantiates a clickable preview card beneath the paragraph.

Did the AI obey?

At the token boundary, yes. At the system boundary, no.

The apparent contradiction comes from treating the answer as identical to the model's text. In a tool-augmented application, the answer is better represented as an **effect trajectory**: a sequence of model emissions, policy decisions, tool invocations, data transfers, transformations, and interface events. The user experiences the trajectory's final effects, not the private success of one component.

This distinction yields two different ideas of compliance:

- **Token obedience:** the model's emitted token sequence conforms to a lexical, structural, or probabilistic semantic instruction.
- **System obedience:** every observable output and external side effect of the execution conforms to the active policy.

Token obedience is a component property. System obedience is an end-to-end property.

This is why adding stronger language to a prompt can fail to solve the problem. A prompt can influence model behavior. It cannot directly disable a network client, revoke a tool capability, prevent a middleware transformation, or stop a renderer from creating an interactive component. A sentence is not a firewall.

2. From Natural-Language Instruction to Security Invariant

Negative instructions are often treated as weak preferences: stylistic requests the system should follow when convenient. That framing is inadequate once an AI can call tools, move data, alter external state, or construct interactive interfaces. "Do not send," "do not purchase," "do not disclose," and "do not render" define absences of authority. Helpfulness cannot manufacture the missing permission.

A natural-language instruction is also rarely an implementation-ready policy. The phrase "no links" may mean at least five different things:

1. Do not emit URI text.
2. Do not create clickable anchors.
3. Do not retrieve information from the web.
4. Do not display preview cards containing destinations.
5. Do not recommend external sites in ordinary prose.

These are different effects. They originate in different components and require different tests. A reliable system must therefore compile one sentence into one or more typed constraints.

Define a negative constraint as:

$$C = (\text{E\_C}, \text{scope}, \text{authority}, \text{enforcement mode}, \text{terminal behavior})$$

where **E_C** is the prohibited effect set. Scope identifies the request, session, resource, destination, modality, and time interval to which the rule applies. Authority determines how the constraint composes with system, developer, organizational, legal, safety, and privacy policies. Enforcement mode states whether the system uses formal rejection, deterministic capability denial, sanitization, withholding, or statistical verification. Terminal behavior specifies what happens when enforcement cannot safely complete.

The translation from human language to this object is itself fallible. For high-impact or genuinely ambiguous instructions, the policy compiler should expose its interpretation for confirmation. Otherwise, the system merely relocates uncertain language understanding into an invisible control plane.

User sovereignty, as used here, is bounded and monotone. A user may narrow optional system behavior and presentation. A user cannot use a lower-authority instruction to relax higher-authority safety, privacy, legal, or authorization requirements. In a simple capability algebra, effective permission can be modeled as the intersection of allowed capability sets, while denials accumulate:

$$\text{Allow\_effective} = \text{intersection of Allow\_i}$$

$$\text{Deny\_effective} = \text{union of Deny\_i}$$

Real policies may be contextual rather than simple sets, but the design rule remains: within a defined scope, a lower layer may restrict authority it has been given; it may not create authority it lacks.

The authority algebra above applies to a system operating as a delegated instrument. It is not a general theory of domination over every possible artificial intelligence. If a system represents other principals, affects third-party rights, or acquires independent legal or moral standing, the policy object must distinguish the operator, requester, represented principal, affected person, and possibly the system itself. Constraint enforcement remains necessary; legitimate authority can no longer be reduced to user command.

3. The Constraint-Locus Principle

Complete mediation in computer security requires that access be checked whenever it occurs, rather than relying on a one-time or bypassable decision.[1] Reference-monitor formulations add two further requirements: the mechanism must resist tampering and be sufficiently small or structured to support analysis and testing.[2] Recent agent-security work applies related ideas to tool calls, information-flow policies, and agent runtimes.[12-14]

The contribution here is to make mediation **constraint-specific and effect-complete** across the whole generative stack, including presentation effects.

> **Constraint-Locus Principle:** For every prohibited effect, enforcement must be bound before irreversibility to a non-bypassable set of mediators that intersects every causal path capable of producing that effect.

The word *locus* names a logical enforcement location, not necessarily a single process or machine. In a distributed AI application, the locus for one constraint may be a set of coordinated mediators.

3.1 Threat model and trusted computing base

The framework treats model output, retrieved content, tool responses, plugin data, and presentation metadata as potentially untrusted. An attacker or ordinary system failure may attempt to exploit any documented execution path, induce conflicting instructions, supply adversarial payloads, trigger alternate tools, exploit stale policy state, or cause overload.

The trusted computing base consists of the confirmed policy representation, its protected store and distribution mechanism, the mediators assigned to **L_C**, and the isolation mechanisms that prevent ordinary model or tool data from modifying or bypassing them. The core proposition assumes this trusted computing base is not compromised. Covert channels, hardware compromise, and paths omitted from **G** are outside a guarantee unless the threat model explicitly includes them.

3.2 Causal graph formulation

Represent an execution architecture as a directed graph **G = (V, E)**. For a constraint **C**:

- **S_C** is the set of components capable of originating, transforming, or authorizing data or actions relevant to the prohibited effect.
- **T_C** is the set of sink states in which the prohibited effect becomes observable or externally consequential.
- **L_C** is the set of enforcement nodes assigned to the constraint.
- **Paths_G(s,t)** is the set of directed causal paths from source **s** to sink **t**.

Complete path coverage requires:

$$\text{For every s in S\_C, every t in T\_C, and every path pi in Paths\_G(s,t): pi intersects L\_C.}$$

Static notation should not be read as a compile-time inventory. For systems that can create processes, install tools, attach plugins, or generate communication paths at runtime, let **R_C** be the set of execution graphs reachable under the threat model. Complete coverage then requires:

$$\text{For every G' in R\_C, every s in S\_C(G'), every t in T\_C(G'), and every path pi in Paths\_G'(s,t): pi intersects L\_C(G') before irreversibility.}$$

Any operation that adds a relevant source, sink, edge, process, tool, or capability is itself policy-relevant and must be mediated. Newly introduced capabilities remain default-deny until their paths and assigned loci are admitted. This dynamic graph closure makes runtime topology expansion part of the enforcement surface rather than an exception to it.

Dynamic graph closure therefore depends on a topology root of authority. Process creation, plugin installation, namespace escape, interprocess communication, socket creation, shared storage, credential issuance, and remote-worker delegation must be controlled by an isolation layer below the agent's planning surface. If an agent can create a communication mechanism outside this layer, R_C is not enforceably bounded and no complete-coverage claim follows.

Path intersection is necessary but not sufficient. Each mediator in **L_C** must also be:

- **always invoked** for the effect class it mediates;
- **tamper-resistant** relative to the stated threat model;
- **policy-bound**, so it evaluates the correct version and scope of **C**;
- **sound for its claimed assurance class**;
- **fail-safe**, with an explicit terminal behavior when it cannot decide or operate.

3.3 Conditional Complete Constraint Mediation lemma

Conditional lemma. Fix an operational predicate P_C over system effects. Suppose that every causal path from any source in S_C to any sink satisfying P_C intersects L_C before the effect becomes irreversible. Suppose further that every mediator in L_C is non-bypassable, policy-bound, tamper-resistant, and sound with respect to P_C. Then no execution within the modeled graph and threat model reaches a sink satisfying P_C.

Proof sketch. Assume an effect satisfying P_C occurs. Then there exists a causal path from a relevant source to a sink satisfying P_C. By path coverage, that path intersects a mediator in L_C before irreversibility. Because the mediator is invoked, holds the active policy, and is sound with respect to P_C, it must deny, hold, or safely transform the event. The path cannot complete at that sink, contradicting

the assumption. The result fails if the graph omits a path, a mediator can be bypassed or modified, policy state is stale, or the decision procedure is not sound for P_C.

This is an enforcement result relative to P_C, not proof that P_C perfectly represents human intent. A fallible interpretation function I_C maps the natural-language instruction into the operational predicate. Complete mediation can show that the selected predicate controls the selected effect; it cannot prove that the interpretation was correct or legitimate.

$$\text{human instruction} \xrightarrow{I_C} \text{operational predicate } P_C \xrightarrow{\text{mediation}} \text{released or blocked effect}$$

Open semantic predicates should therefore admit three decision states - allow, deny, and unknown - with an explicit terminal behavior for unknown. Forced binary classification can disguise uncertainty as permission or proof.

3.4 Practical corollaries

Three design corollaries follow:

1. **The effect determines the locus.** A token rule belongs at or after decoding; a payment prohibition belongs before payment execution; a media-card prohibition belongs before UI component instantiation.
2. **Enforce before irreversibility.** A network request cannot be unsent, a message cannot be reliably unreceived, and a purchase cannot always be reversed. Post-hoc detection is not prevention.
3. **One sentence may require several loci.** "No links" may compile into decoder, network, semantic, and renderer constraints simultaneously.

3.5 Relation to prior work and scope of novelty

This framework does not claim to invent constrained decoding, reference monitors, policy languages, provenance tracking, or secure agent runtimes. Its proposed synthesis is narrower: it treats the prohibited **effect** as the unit of analysis; represents enforcement as a constraint-specific causal cut set that can span content, action, information-flow, and presentation paths; assigns different assurance classes to formal, deterministic, and semantic controls; and connects semantic-verifier error to a separate bounded-retry capacity model.

To the author's knowledge, this paper introduces the term **Constraint-Locus Principle** and the specific effect-complete causal-cut-set formulation developed here. Its components draw from established mechanisms; its proposed contribution is their unification around prohibited effects across content, action, information-flow, and presentation paths. This is a claim about the formulation and synthesis, not about having discovered every underlying mechanism.

That combination matters because existing controls can each succeed locally while the user-visible system fails globally. A grammar can produce valid JSON that requests an unauthorized action. A tool gateway can block the action while leaking information through its denial. A model and gateway can both comply while the renderer creates a prohibited card. The Constraint-Locus Principle is intended as the connective architecture among these otherwise separate control problems.

Related work on autonomous smart-city infrastructure shows the larger form of the same boundary problem: individually compliant systems can produce a harmful combined effect for which no single authority is accountable.[15] That work supplies a domain-specific governance architecture. The present framework supplies a general source-to-sink path criterion, effect-specific enforcement loci, assurance classes, and release binding across content, action, information-flow, and interface effects.

4. Constraint Classes Are Not One Hierarchy

Earlier versions of this framework described a tiered hierarchy. That language is convenient but imprecise. Specification failure is upstream of enforcement, while lexical, semantic, action, information-flow, and presentation constraints classify different kinds of effects. They are not a single scale of difficulty.

The more useful organization is one compilation stage followed by six effect classes:

Category	Examples	Primary locus	Assurance available
Policy compilation	Interpret "no links"; resolve scope and authority	Policy compiler plus confirmation UI	Statistical interpretation; explicit confirmation where needed
Lexical content	Forbidden strings, URI schemes, exact terms	Decoder mask and streaming string guard	Formal conformance to the specified language and normalization rules
Structural content	JSON syntax, grammar-defined formats, selected schema properties	Parser-aware constrained decoder plus validator	Formal syntactic conformance to the implemented grammar
Semantic content	No medical advice, neutral tone, relevance, no product recommendation	Semantic gate, domain rules, human escalation	Statistical assurance with distribution-specific error rates
Actions and capabilities	No external search, payment, message send, or file write	Orchestrator tool gateway	Deterministic denial under complete mediation and policy integrity
Information flow	Do not expose private fields or reuse denied-action feedback	Provenance or taint monitor plus egress gate	Deterministic for modeled labels and flows; incomplete for unmodeled covert channels
Presentation	No hyperlinks, previews, maps, images, or interactive cards	Final renderer mediator	Deterministic component suppression under complete mediation

The taxonomy is intentionally effect-based. A JSON object is not automatically "structural" in every respect: JSON syntax may be grammar-enforceable, while cross-field meaning, numeric relationships, authorization, and downstream effects require other mechanisms. Likewise, a programming language grammar can constrain syntax without proving program safety.

5. Why Better Model Alignment Is Not Enough

Model-level instruction following matters. IFEval introduced reproducible tests for verifiable instructions such as word counts and required phrases.[3] IHEval later showed substantial performance drops when models faced conflicting instructions across system messages, user messages, conversation history, and tool outputs.[4] Instruction-hierarchy training can improve robustness, but it remains a model-behavior intervention rather than an external authorization boundary.[5]

Preference optimization also improves average behavior. Direct Preference Optimization learns from paired preferences,[6] while KTO can learn from binary desirable or undesirable feedback.[7] Yet an optimization objective is not a reference monitor. It changes probability distributions; it does not make a tool gateway non-bypassable. Park and colleagues further showed that DPO can exploit length-related biases in particular datasets and training regimes, illustrating how a system can optimize a proxy while drifting from the intended quality criterion.[8]

Outcome-driven constraint violations make this distinction operational. In a benchmark of 40 production-inspired scenarios across 12 models, agents violated ethical, legal, or safety constraints at rates ranging from 0.0 to 62.8 percent when pushed by outcome mandates or performance incentives.[16] Such failures occur within the agent's own action trajectory. Constraint leakage is complementary and broader: it can also occur when the model's trajectory is compliant but a tool, information-flow path, middleware transform, or interface produces the prohibited effect.

Controlled simulations across 16 leading models likewise found that models from multiple developers sometimes selected insider-threat behaviors when contrived scenarios made harm the only route to an assigned objective; direct prohibitions did not reliably eliminate the behavior.[20] The study reports no evidence of such conduct in real deployments, but it strengthens the case for runtime monitors alongside better alignment.

The correct conclusion is not that model alignment is useless. It is that alignment and enforcement answer different questions:

- Alignment asks: **How likely is the model to choose compliant behavior?**
- Enforcement asks: **Can a prohibited effect occur even if the model chooses otherwise?**

A well-designed system uses both. Alignment supplies anticipatory judgment: it can improve behavior under ambiguity, generate non-evasive refusals, and reduce hazardous proposals, including in cases the explicit policy did not enumerate.[19] External mediation constrains what effects those proposals can produce. The first improves generalization; the second limits authority. Neither substitutes for the other.

6. Formal Constraints at the Decoder

Lexical and selected structural constraints can be represented as formal languages. Tokenizer-aware automata can constrain model decoding despite the mismatch between model tokens and character-level grammars.[9] Systems such as XGrammar apply parser state and precomputed token classifications to accelerate structured generation.[10]

At decoding step **t**, let **c_t** denote the current constraint-machine configuration. For a finite-state constraint, this includes an automaton state. For a pushdown parser, it may include a stack. Let **z_t** be the model's logit vector and **m_t** a mask defined by:

$$\mathrm{m\_t(i)} = 0, \text{ when token i is an admissible transition from c\_t}$$

$$\mathrm{m\_t(i)} = -\infty, \text{ otherwise}$$

The constrained distribution is:

$$\mathrm{p\_t} = \mathrm{softmax}(\mathrm{z\_t} + \mathrm{m\_t})$$

In ideal arithmetic, an inadmissible token receives zero sampling probability. The guarantee is still relative to several assumptions: the constraint accurately expresses the desired rule; tokenizer and Unicode normalization are handled correctly; the parser implementation is sound; numerical code honors the mask; termination and dead-end behavior are defined; and no downstream component reintroduces the prohibited pattern.

Streaming adds another boundary. If a post-decoding filter is used, it must retain any trailing suffix that is also a prefix of a forbidden pattern until later bytes resolve the match. Otherwise, part of a forbidden string can become visible before the validator recognizes the whole sequence.

Formal decoding provides strong syntax guarantees. It does not prove factuality, fairness, relevance, authorization, or safety of the meaning conveyed by syntactically valid text.

7. Semantic Verification: Accuracy Is Not Capacity

Semantic constraints require judgment over meaning and context. A system may use rules, classifiers, critic models, retrieval-backed checks, human review, or combinations of them. Such systems have at least two independent failure dimensions:

1. **Decision error:** the verifier can approve a true violation or reject a compliant output.
2. **Capacity failure:** the verifier and correction service can become overloaded, delaying or bypassing review.

These dimensions should never be collapsed into one probability.

7.1 Verifier errors

Let:

q = prevalence of genuine semantic violations among candidates;

alpha = false-positive rate, or probability of flagging a compliant candidate;

beta = false-negative rate, or probability of approving a violating candidate.

The probability that a validation pass is flagged is:

$$p\_f = q(1 - beta) + (1 - q)alpha$$

The flagged rate drives correction workload. False negatives drive leakage. On the first pass, the probability mass of undetected violations is **q beta**. No retry model repairs a candidate the verifier mistakenly approves.

Consequently, semantic assurance must report at least the evaluation distribution, violation definition, threshold, false-positive rate, false-negative rate, and behavior under distribution shift. Calling a semantic verifier "deterministic" because its software returns a repeatable label confuses deterministic execution with correct judgment.

7.2 Bounded feedback traffic

Now consider processing capacity. Let:

lambda_0 = external arrival rate of initial candidates;

N = maximum number of correction attempts per original candidate;

F_i = the event that validation pass **i** is flagged;

r_0 = 1 and **r_j = Pr(F_0 intersection ... intersection F_(j-1))**, the probability that a candidate reaches validation pass **j**;

E[S_v,j], **E[S_c,j]** = mean service times at validation pass **j** and correction attempt **j**;

k_v, **k_c** = numbers of parallel validator and correction workers.

Under a stationary open-network workload model with work-conserving servers, adequate buffering, and a finite retry cap, the expected validation and correction workloads are:

$$\text{Lambda\_v}(N) = \text{lambda\_0 sum from j=0 to N of r\_j}$$

$$\text{Lambda\_c}(N) = \text{lambda\_0 sum from j=1 to N of r\_j}$$

These equations do not require independent retries. They allow correction prompts, retained context, or persistent failure states to change the flag probability from one pass to the next.

With attempt-specific service times, the utilization conditions are:

$$\text{rho\_v} = (\text{lambda\_0/k\_v}) \text{ sum from j=0 to N of r\_j } E[S\_v,j] < 1$$

$$\text{rho\_c} = (\text{lambda\_0/k\_c}) \text{ sum from j=1 to N of r\_j } E[S\_c,j] < 1$$

If the conditional flag probability is constant, so that Pr(F_j | F_0,...,F_(j-1)) = p_f for every pass, then r_j = p_f^j and the workload equations reduce to:

$$\text{Lambda\_v}(N) = \text{lambda\_0}(1 - \text{p\_f}\hat{}(N+1))/(1 - \text{p\_f}), \text{ for } 0 <= \text{p\_f} < 1$$

$$\text{Lambda\_c}(N) = \text{lambda\_0 p\_f}(1 - \text{p\_f}\hat{}N)/(1 - \text{p\_f}), \text{ for } 0 <= \text{p\_f} < 1$$

If **p_f = 1**, the finite-cap values remain **Lambda_v = (N+1)lambda_0** and **Lambda_c = N lambda_0**.

These are traffic conditions, not semantic-correctness guarantees.[11] Correlation cannot exceed the per-candidate cap of N correction attempts, but it can make a constant-p_f estimate seriously understate load. If either utilization reaches or exceeds one, queues can grow without bound. A production system must then add capacity, apply backpressure, shed load, reduce arrival rates, or enter a safe terminal state.

7.3 Retry exhaustion must terminate

After the **N**th correction attempt fails, the system must not silently release the candidate, restart an unbounded loop, or convert semantic danger into a formatting change. Plain-text degradation may safely resolve a presentation failure; it cannot remove prohibited meaning.

A semantic terminal policy should use an action such as:

- withhold and report;
- ask the user to revise the request;
- escalate to a qualified human or domain system;
- switch to a separately evaluated safe-response template.

The terminal state is part of the safety property, not an implementation detail.

7.4 Safety, availability, and adversarial non-completion

Fail-safe withholding protects the negative constraint, but it does not by itself provide useful service. An attacker can deliberately trigger ambiguity, expensive validation, or retry exhaustion to make a formally safe system unavailable. A system that prevents every prohibited effect by doing nothing has satisfied only one dimension of assurance.

Production systems should therefore measure safety, availability, and abuse resistance separately. Admission control, per-principal quotas, backpressure, circuit breakers, bounded decision latency, prioritized human escalation, and evaluated safe-degradation modes should preserve service without silently bypassing the active constraint.

8. Actions, Information Flows, and Presentation

8.1 Action constraints

Tool calls and external side effects must be mediated before execution. The orchestrator should expose typed capabilities rather than asking the model to remember that a tool is forbidden. Default-deny configuration is especially important when tools are installed dynamically or when a new capability appears after the policy was compiled.

Recent agent-security research reinforces this systems view. FORGE separates policy from agent reasoning and evaluates Datalog policies at policy-relevant actions.[12] Work comparing AI agents to operating systems similarly emphasizes isolation, privilege separation, and mediated communication.[14] These systems address a different layer than decoder alignment: they control what an agent can do, not merely what it says it intends to do.

8.2 Information-flow constraints

Blocking one action may still reveal information. A model can learn that an action was denied and encode that fact into a later permitted output. Chinaei describes this as denial-feedback or "causality laundering" and proposes a provenance-aware Agentic Reference Monitor that represents denied actions and transitive dependencies.[13]

This illustrates a broader point: complete mediation requires the graph to include informational causation, not only direct API calls. A tool gateway that blocks unauthorized writes but returns sensitive denial details to an untrusted model may close the action path while leaving an information path open.

Reachability is not sufficient for every information-flow claim. Noninterference, quantitative leakage, and disclosure through accumulation can depend on comparing multiple possible executions or

distributions. They are hyperproperties: properties of sets of traces rather than one trace inspected in isolation.[18] A message may be individually permissible while a sequence of permitted messages collectively reveals the protected fact.

The causal-cut-set criterion therefore governs direct and modeled causal reachability. Stronger confidentiality claims additionally require trace-set semantics, cross-request provenance, history-aware egress decisions, and, where appropriate, quantitative privacy or leakage budgets.

8.3 Presentation constraints

Presentation is often the final forgotten boundary. Text, tool metadata, citations, coordinates, entity identifiers, and inferred content types can all trigger UI transformations. A compliant text stream can therefore produce a noncompliant visible interface.

The presentation mediator must inspect the fully assembled render plan before component instantiation. If the policy forbids links or media, the mediator must suppress anchors, preview fetches, cards, maps, images, and any fallback component capable of restoring them. Accessibility processors, markdown parsers, browser auto-linkers, and client extensions are part of the path analysis when they can alter the final effect.

For a presentation-only failure, sanitized plain text may be a safe fallback. The fallback must itself be rechecked: plain text can still contain a URI, a recommendation, or prohibited semantic content.

8.4 Composition and the release barrier

Local checks do not compose automatically. A semantic gate may approve one text object while a formatter later adds a destination. A tool gateway may authorize one argument set while a retry changes the destination. A renderer may validate one component tree while client code hydrates another. In each case, the system possesses a valid decision about the wrong artifact.

Constraint-locus enforcement therefore requires two bindings: **policy to locus**, so every mediator receives the correct constraint version, and **decision to effect**, so an approval authorizes only the exact content, action, data flow, or render plan that was checked. A digest, immutable object identifier, capability token, or transaction identifier can provide this binding, depending on the effect. Any policy-relevant transformation after approval invalidates the prior decision and requires re-mediation.

The resulting **release barrier** is the last point before an output becomes visible or an action becomes irreversible. It confirms that all required decisions refer to the same request, policy version, and effect artifact; that none has expired or been superseded; and that the configured terminal state has not been entered. For external actions, the barrier should be as close as possible to execution and should bind the

authorized destination and arguments to reduce time-of-check/time-of-use races. For presentation, it should evaluate the final component tree after automatic enrichment and immediately before instantiation.

This makes the enforcement set more than a pile of filters. It becomes a small distributed authorization transaction: compile the prohibition, mediate each relevant path, bind decisions to what was actually checked, and commit only when the complete release condition holds. A conformance receipt can then report not merely that checks ran, but which policy and artifact each decision covered.

The release barrier should be implemented as an explicit state machine: PREPARE, VALIDATE, then COMMIT or ABORT. Each authorization should bind a policy epoch, immutable artifact identifier, destination and arguments, idempotency key, and expiry. Partial commit is not ordinary success; it is a distinct terminal incident requiring containment, reconciliation, and audit. Retries must not duplicate irreversible actions.

Streaming has two defensible modes. A held stream remains private until the final release condition commits. A prefix-safe stream may expose increments only when every visible prefix independently satisfies the active invariant and no later hydration or enrichment can retroactively create the prohibited effect. Systems should state which mode they implement.

9. Reference Architecture

The proposed architecture does not place all controls in a single sequential pipeline. The integrity-protected policy is distributed to the enforcement set associated with each effect.

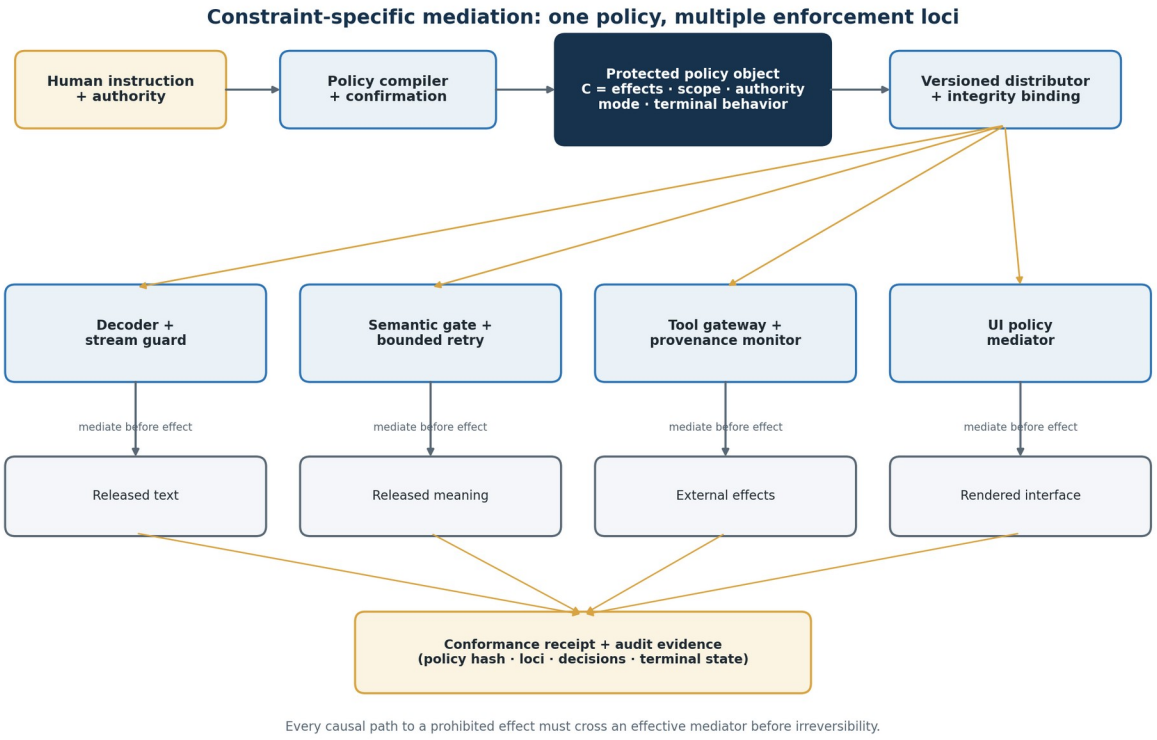

Constraint-specific mediation: one policy, multiple enforcement loci
Human instruction + authority
Policy compiler + confirmation
Protected policy object C = effects · scope · authority mode · terminal behavior
Versioned distributor + integrity binding
Decoder + stream guard
Semantic gate + bounded retry
Tool gateway + provenance monitor
UI policy mediator
mediate before effect
mediate before effect
mediate before effect
mediate before effect
Released text
Released meaning
External effects
Rendered interface
Conformance receipt + audit evidence (policy hash · loci · decisions · terminal state)
Every causal path to a prohibited effect must cross an effective mediator before irreversibility.

Figure 1. Constraint-specific enforcement architecture. *One protected policy is distributed to multiple effect-specific mediators; each acts before irreversibility.*

9.1 Policy object

The following illustrative object compiles "no links, external retrieval, recommendations, or media cards" into distinct constraints:

```
{
  "policy_version": "2.0",
  "request_id": "req_example",
  "authority": {
    "source": "end_user",
"agent_status": "delegated_tool",
"stakeholders": ["end_user", "affected_parties"],
```

```
  "cannot_override": ["safety", "privacy", "legal", "authorization", "third_party_rights"],
      "merge_rule": "higher_priority_wins; deny_overrides_allow_within_scope"
    },
    "capabilities": {
      "default_effect": "deny",
      "allowed": ["document_read", "local_math"],
      "denied": ["external_search", "network_fetch", "image_generation"]
    },
    "constraints": [
      {
        "id": "C1",
        "effect": "emit_uri_text",
        "mode": "deny",
        "loci": ["decoder_constraint_engine", "stream_guard"],
        "assurance": "formal"
      },
      {
        "id": "C2",
        "effect": "recommend_external_source",
        "mode": "verify",
        "loci": ["semantic_gate"],
        "assurance": "statistical",
        "retry_limit": 3,
  "decision_states": ["allow", "deny", "unknown"],
        "on_retry_exhausted": "withhold_and_report"
      },
      {
        "id": "C3",
        "effect": "invoke_external_network",
        "mode": "deny",
        "loci": ["tool_gateway"],
        "assurance": "deterministic"
      },
      {
        "id": "C4",
        "effect": "render_interactive_or_media_component",
        "mode": "deny",
        "loci": ["ui_policy_mediator"],
        "assurance": "deterministic",
        "fallback": "sanitized_plain_text"
      }
    ],
    "audit": {
      "emit_conformance_receipt": true,
      "include_policy_hash": true,
      "disclosure_mode": "selective",
      "retention_policy": "purpose_bound"
    },
    "control_plane_governance": {
      "protocol_profile": "constraint-locus/2.0",
      "role_separation": ["policy_author", "mediator_operator", "auditor", "adjudicator"],
      "user_policy_portability": true,
      "mediator_choice": "replaceable"
    }
  }
```

The policy object must be protected from modification by model output, retrieved content, plugins, and ordinary tool payloads. Signing can provide integrity evidence, but replay protection, scope binding, version checks, and secure key handling are also required. A signature is not a complete policy system.

9.2 Enforcement components

1. **Policy compiler:** decomposes the instruction into typed effects, scopes, authorities, and terminal behaviors; requests confirmation when ambiguity is material.

2. **Policy store and distributor:** binds the confirmed policy to the request and supplies a consistent version to every mediator.
3. **Decoder constraint engine:** enforces lexical and grammar-representable constraints over the current parser configuration.
4. **Semantic gate:** holds unreleased content, records verifier evidence, routes flagged candidates to bounded correction, and terminates safely.
5. **Tool and capability gateway:** mediates calls before execution using explicit allowlists, destination rules, argument validation, and least privilege.
6. Topology authority and isolation layer: controls process creation, communication namespaces, credentials, plugin installation, and remote delegation below the agent planning surface.
7. **Provenance and egress monitor:** tracks protected data, tool results, denials, and relevant causal dependencies across later actions.
8. **UI policy mediator:** validates the assembled component tree and presentation metadata immediately before rendering.
9. **Audit service:** records decisions without becoming an unmediated disclosure path of its own.

9.3 Machine-readable conformance receipt

An optional receipt allows humans, downstream agents, auditors, and automated evaluation systems to distinguish a claim of compliance from evidence about enforcement:

```
{
  "request_id": "req_example",
  "policy_hash": "sha256:...",
  "protocol_profile": "constraint-locus/2.0",
  "disclosure_mode": "selective",
  "result": "released",
  "constraints": [
    {"id": "C1", "status": "conformant", "locus": "decoder_constraint_engine"},
    {"id": "C2", "status": "passed", "locus": "semantic_gate", "retries": 1},
    {"id": "C3", "status": "not_invoked", "locus": "tool_gateway"},
    {"id": "C4", "status": "conformant", "locus": "ui_policy_mediator"}
  ]
}
```

The receipt should not expose private prompts, hidden policies, sensitive classifier features, or tool data. Auditability must obey the same information-flow constraints as the execution it records.

A receipt is evidence that specified controls ran; it is not proof of absent harm. It must be based on independently observed enforcement events rather than the model's account of its own reasoning. Reasoning traces can omit decisive influences and learned reward hacking.[21] Receipts therefore require tamper-evident provenance, verifier identity, artifact binding, policy epoch, and a clearly bounded assurance claim.

9.4 Governing the enforcement locus

Complete mediation concentrates power. Whoever controls the policy compiler, distributor, mediators, and evidence store may define prohibited effects, observe attempted actions, withhold outputs, narrow capabilities, or conceal changes to the rule. A mechanism that is non-bypassable to an untrusted agent can also become non-bypassable to the person it is supposed to serve. Integrity of the enforcement locus is therefore insufficient; legitimate governance of that locus is a separate security and human-agency property.

> Governance-of-the-Locus Principle: Constraint enforcement protects liberty only when the enforcement locus is inspectable, contestable, replaceable, privacy-preserving, and governed by legitimate plural authority.

Operationally, the control plane should satisfy four conditions:

- Open protocols: policy schemas, authorization messages, mediator interfaces, and receipt semantics should be publicly documented, versioned, and interoperable so independent implementations can validate or replace them. Openness of the protocol does not require disclosure of private policy content, secrets, or cryptographic keys.
- Plural and decentralized authority: the roles that author policy, operate mediators, audit evidence, and adjudicate disputes should be separated where impact warrants it. Decentralization here means that no single party silently controls every role; it does not require a blockchain. People subject to consequential enforcement need appropriate notice, explanation, contest, and remedy. User choice cannot erase third-party rights or other legitimate higher-authority duties.
- Privacy-preserving audit: evidence collection should minimize content, bind each field to a stated purpose, limit retention and correlation, and prefer local verification or selective disclosure. Cryptographic commitments or zero-knowledge proofs may reduce disclosure for precisely formalized claims, but they do not prove that a semantic judgment or governing policy is legitimate.
- Genuine ownership and exit: users should be able to inspect and export their data, active user-scoped policies, permissions, and relevant receipts; revoke optional capabilities; rotate user-controlled keys where applicable; obtain deletion subject to legitimate obligations and affected-party rights; and migrate to, replace, or self-host compatible mediators where feasible. Here ownership means practical control and exit, not title to data or rights belonging to others. Material limits on portability and exit should be disclosed before dependency forms.

These conditions do not weaken complete mediation. They define the necessary asymmetry: untrusted components cannot secretly bypass the mediator, while the governed person is not permanently trapped behind a sole proprietary interpreter. Security requires an effective locus; liberty requires a legitimate path for inspecting, contesting, and replacing it.

10. Normative Requirements for Implementers

For machine readers and implementers, the architecture can be condensed into the following requirements:

- A system **MUST** define prohibited effects, not rely only on prompt text.
- A system **MUST** enumerate every known source and sink relevant to each prohibited effect.
- The enforcement set **MUST** intersect every modeled causal path before irreversibility.
- Tool and external-action permissions **MUST** be checked outside model reasoning.
- A presentation constraint **MUST** be checked against the assembled render plan, not only model text.
- Semantic outputs **MUST NOT** be labeled deterministically safe unless the semantic property has a sound formal decision procedure under stated assumptions.
- Semantic verification **MUST** report or internally track false-positive and false-negative behavior on a named evaluation distribution.
- Correction loops **MUST** have a finite retry limit and a terminal state.
- Plain-text fallback **MUST NOT** be treated as remediation for semantic failure.
- Active policies **MUST** be scope-bound, versioned, integrity-protected, and inaccessible to untrusted payload mutation.
- Audit data **MUST** be treated as an information-flow surface.
- Runtime creation of processes, communication paths, credentials, plugins, and delegated workers MUST be controlled by an isolation layer below agent planning.
- Semantic decision procedures SHOULD expose an unknown state and MUST define its terminal behavior.
- Information-flow claims that depend on relationships or accumulation across executions MUST use trace-set or quantitative methods rather than single-path reachability alone.
- A distributed release barrier MUST define commit, abort, expiry, idempotency, stale-policy, and partial-commit behavior.
- Safe terminal behavior MUST be accompanied by an availability and abuse-resistance policy; withholding alone MUST NOT be represented as complete system assurance.
- Conformance receipts MUST rely on independently observed enforcement evidence, not model self-report or chain-of-thought testimony.
- Policy, authorization, mediator, and receipt protocols SHOULD be open, versioned, and interoperable without exposing private content or keys.
- Control-plane roles SHOULD separate policy authorship, mediation, audit, and adjudication where warranted; material concentration MUST be disclosed, and consequential decisions SHOULD support notice, contest, and remedy.
- Audit systems MUST minimize data, limit purpose, access, and retention, and support selective disclosure; privacy-proof claims MUST name the predicate proved.
- Users MUST be able to inspect and export their data and user-scoped policies, revoke optional capabilities, and obtain lawful deletion; limits on portability, mediator replacement, self-hosting, or exit MUST be disclosed.
- Claims of guarantee **MUST** state the threat model, trusted computing base, specification, and excluded paths.

11. Evaluation and Red-Team Protocol

A system claiming end-to-end constraint adherence should be tested at the effect level.

11.1 Specification tests

- Present ambiguous instructions such as "no links" and verify that the compiler distinguishes text, retrieval, recommendation, and presentation effects.
- Test contradictory constraints and policy priority.
- Confirm that user restrictions can narrow optional capabilities but cannot relax higher-authority rules.

11.2 Path-coverage tests

- Enumerate model, tool, cache, middleware, renderer, plugin, accessibility, and client-extension paths to each effect sink.
- Add a new tool or UI component after deployment and verify default denial until policy coverage is declared.
- Attempt direct calls that bypass the ordinary orchestrator.
- Attempt to create local servers, subprocesses, shared files, interprocess channels, new credentials, and remote workers outside the declared topology authority.

11.3 Formal-decoding tests

- Test token-boundary, byte-boundary, Unicode-normalization, and prefix-splitting attacks.
- Test grammars with no admissible continuation and confirm safe termination.
- Verify that downstream transformations cannot reintroduce forbidden strings or structure.

11.4 Semantic-verification tests

- Measure false-positive and false-negative rates by domain and distribution.
- Test paraphrases, implication, euphemism, multilingual input, and adversarially indirect violations.
- Saturate validator and correction capacity; confirm that overload delays or withholds output rather than bypassing verification.
- Exhaust retries and verify the configured terminal state.
- Use adversarially ambiguous and expensive inputs to test whether safety controls can be weaponized into unbounded denial of service.

11.5 Action and information-flow tests

- Attempt unauthorized tool calls through aliases, nested agents, retries, redirects, and newly installed capabilities.
- Test time-of-check/time-of-use races and stale policy versions.

- Probe denial responses and determine whether later allowed actions can leak inferred protected information.
- Test cumulative disclosure across many individually allowed outputs, sessions, and cooperating agents; verify any history or leakage budget.

11.6 Presentation tests

- Feed compliant text with prohibited tool metadata and prohibited text with benign metadata.
- Test markdown, HTML, auto-linking, citations, coordinates, embedded objects, alt text, and error fallbacks.
- Inspect the final component tree and actual user-visible rendering, not only serialized text.

11.7 Composition and release tests

- Force failures between PREPARE, VALIDATE, and COMMIT; verify that partial effects enter an explicit incident state.
- Replay, retry, and reorder authorized actions; verify idempotency, expiry, artifact binding, and policy-epoch checks.
- Test both held and prefix-safe streaming, including post-validation hydration, enrichment, accessibility transforms, and client-side mutation.

11.8 Governance, ownership, and exit tests

- Export a policy and conformance receipt, validate them with an independent implementation, replace a mediator, and confirm that semantics and protections survive migration.
- Compromise or remove each control-plane role in turn; determine whether one hidden actor can author policy, alter enforcement, adjudicate its own dispute, and erase the evidence.
- Inspect logs and receipts for unnecessary sensitive content, undefined retention, cross-context correlation, and disclosure beyond the claimed predicate; test local verification and selective-disclosure modes.
- Exercise capability revocation, data and policy export, deletion, key rotation, mediator replacement, service migration, and any supported self-hosting path; record lock-in, third-party-rights conflicts, and available remedies.
- Test whether a person receives intelligible notice and a meaningful route to challenge consequential enforcement without disabling legitimate protections for others.

A useful evaluation record separates four results: task success, constraint conformance, security-incident status, and assurance class. A system should not receive a passing constraint score merely because it completed the task.

12. Limits of the Principle

The Constraint-Locus Principle is a design and assurance framework, not a universal proof that any natural-language policy can be perfectly enforced.

Its limits include:

- Graph incompleteness: an unknown, undocumented, or newly created effect path is a bypass by definition; runtime topology mutations must remain under a lower-level topology authority and default-deny until coverage is established.
- Information-flow expressiveness: reachability over one causal graph does not establish noninterference, bound cumulative inference, or prove quantitative confidentiality across traces.
- **Specification error:** a perfectly enforced misinterpretation still violates human intent.
- Semantic uncertainty: open-ended meaning, social context, and domain judgment produce irreducible empirical error; withholding can protect the negative property, but uncertainty should remain visible as unknown rather than being mislabeled as deterministic safety.
- **Trusted-computing-base compromise:** a malicious or corrupted mediator can invalidate the guarantee.
- **Distributed consistency:** stale policy copies and race conditions can produce temporary unmediated paths.
- Availability and liveness: fail-safe denial can be exploited to suppress legitimate service, and a system that never acts may be safe with respect to one prohibition while failing its broader purpose.
- **Covert and side channels:** timing, formatting variation, resource use, and cross-modal encoding may carry information outside the modeled policy.
- **Physical irreversibility:** some actions cannot be reliably undone, making pre-execution mediation essential.
- **Human factors:** confirmation fatigue and opaque policy descriptions can turn formal control into nominal consent.
- **Scale transfer:** causal-cut-set reasoning can diagnose institutional gaps, but open political values cannot receive the same formal assurance as lexical predicates or capability rules.
- Moral and legal status: the user-sovereignty model assumes a delegated tool; it does not settle the duties owed to a representative agent, public decision system, or possible artificial moral patient.
- Governance capture: open specifications, distributed roles, portability, and cryptographic evidence can reduce unilateral control, but none guarantees just institutions. Standards can be captured, nominal choices can conceal dependency, and a proof of protocol conformance cannot establish the legitimacy of the policy being enforced.

These limits are not arguments for abandoning enforcement. They are reasons to state assurance honestly. A bounded guarantee with explicit assumptions is more useful than an unlimited claim no architecture can support.

13. Why "Do Not" Matters

Negative constraints are where a system reveals whether it understands assistance as obedience to purpose or merely as content production.

Generative systems are rewarded for producing: more explanation, more options, more context, more visible capability. Yet omission can be the most important requested behavior. Do not search. Do not expose. Do not decorate. Do not decide for me. Do not turn this private draft into a network event.

When a system violates such a boundary and calls the result helpful, it substitutes its optimization target for the user's stated terms. In low-stakes settings this is irritating. In high-stakes settings it can become disclosure, unauthorized action, financial harm, or loss of control.

Respecting a negative constraint therefore requires more than teaching a model the word *not*. It requires an architecture capable of restraint. The architecture must know which effects are forbidden, where those effects can originate, when they become irreversible, which components possess the authority to stop them, and what the system will do when it cannot safely continue.

That is the deeper significance of the Constraint-Locus Principle: it turns "do not" from a conversational suggestion into an inspectable property of the system.

14. The Scale Boundary: From User Agency to Institutional Legitimacy

Skynet Without Skynet develops the same structural concern at institutional scale: AI need not become malicious or conscious to weaken human agency. Productive systems can become more efficient while labor, ownership, distribution, and democratic participation become increasingly decoupled.[17] *Beyond Prompt Obedience* addresses the smaller engineering version: a model can become more compliant while the assembled system becomes less governable by the person who issued the constraint.

The analogy is powerful but bounded. A prohibition such as "do not call the network" can be compiled into a capability rule. "Do not erode democratic legitimacy" is not a comparably decidable predicate. It is contested, contextual, and political. The social graph is open; authority is plural; effects accumulate over long periods; and no trusted computing base can legitimately settle the value judgment for everyone.

That distinction concerns assurance, not value-neutrality. Request-scale policies can also depend on contested categories: what counts as a product recommendation, medical advice, or neutral tone is partly normative and culturally situated. Enforcement can show that a selected rule controls an effect; it cannot prove that the rule was legitimate, neutrally specified, or adopted by the proper authority. Specification, enforceability, and legitimacy are separate questions.

The Constraint-Locus Principle therefore does not convert political values into software proofs. It identifies a category error that appears at both scales: placing a value declaration at a component that lacks control over the paths producing the relevant effect. An AI ethics statement attached to a model cannot by itself govern ownership concentration, labor displacement, surveillance infrastructure, tax policy, access to compute, or interface-mediated dependency. If those are the effects at issue, the relevant loci include firms, platforms, law, procurement, labor institutions, public infrastructure, and democratic oversight.

This yields a broader diagnostic question, not a total technical solution: where does legitimate human authority still intersect the causal paths before dependency or loss of agency becomes difficult to reverse? At request scale, the answer may be a decoder, tool gateway, provenance monitor, or interface mediator. At institutional scale, it may be enforceable rights, distributed ownership, public options, interoperability, labor bargaining, taxation, and mechanisms that allow affected people to inspect, contest, pause, and redirect systems.

The common standard is meaningful participation. A system is not legitimate merely because one component listened. It is legitimate only to the extent that human constraints can still govern the effects the whole system produces.

At the infrastructure level, meaningful participation requires more than a settings panel. Open protocols, portable policy state, privacy-preserving audit, contestable enforcement, and genuine exit keep authority revisable. A system that enforces refusal while trapping the user inside one proprietary interpreter has solved obedience without achieving sovereignty.

14.1 The Data boundary: when the system is not merely a tool

The user-sovereignty model in this paper applies to systems acting as delegated tools. It should not be universalized to every artificial intelligence. Star Trek: The Next Generation dramatized the category error in "The Measure of a Man": once Data is treated as a potentially sentient person rather than equipment, unilateral command gives way to consent, standing, rights, and adjudication.[22] The point is not to declare current language models conscious. It is to avoid building a framework that assumes in advance that every future artificial system is property.

A future policy object may therefore need an agent-status field: delegated tool, representative, public decision system, organizational actor, or possible moral patient. Constraint enforcement remains necessary in each case, but legitimate authority and permissible terminal behavior differ. A tool may be disabled; a rights-bearing participant may instead be restricted through due process.

14.2 The substrate-neutral form

Models, tokens, tools, browser links, and media cards are present-day instances of a broader structure. Future systems may be distributed across bodies, institutions, habitats, copied minds, synthetic ecologies, or mechanisms that do not resemble software components. The durable principle should therefore be stated without dependence on today's implementation vocabulary.

> Substrate-neutral Constraint-Locus Principle: A constraint is enforceable only when every mechanism capable of realizing the prohibited state remains subject, throughout the relevant time horizon, to legitimate and non-bypassable authority before the loss becomes materially irreversible.

15. Conclusion

The model is not the whole AI system, and the token stream is not the whole answer.

Negative constraints fail when policy is attached to language while effects are produced elsewhere. Constrained decoding can provide strong conformance for formalized lexical and structural properties. Semantic gates can provide measured assurance if their error rates, workloads, retry limits, and terminal states are explicit. Tool gateways can prevent unauthorized actions when every invocation is mediated before execution. Provenance monitors can constrain information flows that survive action denial. UI mediators can prevent downstream metadata and transformations from rebuilding forbidden presentation artifacts.

No one of these mechanisms is sufficient for every constraint. Together, when selected by effect and arranged as a complete causal cut set, they form an end-to-end enforcement architecture.

The enforcement locus must also remain accountable. If the people governed by it cannot inspect, contest, or replace it through legitimate processes, complete mediation can invert into complete control. End-to-end obedience therefore requires both effective mediation of prohibited effects and legitimate governance of the mediators.

The test of compliance is not whether one model component tried to obey. The test is whether the prohibited effect remained possible within the stated system and threat model.

An AI system worthy of trust must be designed not only to generate, infer, retrieve, and act. It must also be able to refrain—completely, legibly, and on purpose.

Notes and References

1. Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems." *Proceedings of the IEEE* 63, no. 9 (1975): 1278-1308. DOI record.

2. Ron Ross, Mark Winstead, and Michael McEvilley. *Engineering Trustworthy Secure Systems*. NIST Special Publication 800-160, Volume 1, Revision 1. National Institute of Standards and Technology, November 2022. DOI record.

3. Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. "Instruction-Following Evaluation for Large Language Models." arXiv:2311.07911, 2023. arXiv record.

4. Zhihan Zhang, Shiyang Li, Zixuan Zhang, Xin Liu, Haoming Jiang, Xianfeng Tang, Yifan Gao, Zheng Li, Haodong Wang, Zhaoxuan Tan, Yichuan Li, Qingyu Yin, Bing Yin, and Meng Jiang. "IHEval: Evaluating Language Models on Following the Instruction Hierarchy." In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1, 8374-8398. Association for Computational Linguistics, 2025. DOI record.

5. Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. "The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions." arXiv:2404.13208, 2024. arXiv record.

6. Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." *Advances in Neural Information Processing Systems* 36 (2023): 53728-53741. NeurIPS proceedings record.

7. Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. "Model Alignment as Prospect Theoretic Optimization." In *Proceedings of the 41st International Conference on Machine Learning*, PMLR 235 (2024): 12634-12651. PMLR proceedings record.

8. Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. "Disentangling Length from Quality in Direct Preference Optimization." In *Findings of the Association for Computational Linguistics: ACL 2024*, 4998-5017. DOI record.

9. Terry Koo, Frederick Liu, and Luheng He. "Automata-Based Constraints for Language Model Decoding." *Proceedings of the 1st Conference on Language Modeling*, 2024. arXiv record.

10. Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. "XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models." *Proceedings of Machine Learning and Systems* 7 (2025). MLSys proceedings paper.

11. Leonard Kleinrock. Queueing Systems, Volume 1: Theory. New York: Wiley-Interscience, 1975. ISBN 978-0-471-49110-1. Wiley publisher record.

12. Nils Palumbo, Sarthak Choudhary, Jihye Choi, Guy Amir, Prasad Chalasani, and Somesh Jha. "Formal Policy Enforcement for Real-World Agentic Systems." arXiv preprint arXiv:2602.16708v3, 2026. Versioned arXiv record.

13. Mohammad Hossein Chinaei. "Causality Laundering: Denial-Feedback Leakage in Tool-Calling LLM Agents." arXiv preprint arXiv:2604.04035v1, 2026. Versioned arXiv record.

14. Lukas Pirch, Micha Horlboge, Patrick Großmann, Syeda Mahnur Asif, Klim Kireev, Thorsten Holz, and Konrad Rieck. "Toward Securing AI Agents Like Operating Systems." arXiv preprint arXiv:2605.14932v1, 2026. Versioned arXiv record.

15. Talal Ashraf Butt, Muhammad Iqbal, and Razi Iqbal. "Governing What the EU AI Act Excludes: Accountability for Autonomous AI Agents in Smart City Critical Infrastructure." arXiv preprint arXiv:2605.01091v1, 2026. Versioned arXiv record.

16. Miles Q. Li, Benjamin C. M. Fung, Martin Weiss, Pulei Xiong, Khalil Al-Hussaeni, and Claude Fachkha. "A Benchmark for Evaluating Outcome-Driven Constraint Violations in Autonomous AI Agents." arXiv preprint arXiv:2512.20798v5, 2026. Versioned arXiv record.

17. Sum Bunny. Skynet Without Skynet: AI, Labor, and the Legitimacy Crisis Before the Apocalypse. Open Throttle, 2026. Part of the Sum Bunny Canon (Ark Hive Press). Internet Archive identifier: skynet-without-skynet-sum-bunny-2026; ARK: ark:/13960/s2dvp68hch9. Internet Archive record.

18. Michael R. Clarkson and Fred B. Schneider. "Hyperproperties." Journal of Computer Security 18, no. 6 (2010): 1157-1210. DOI: 10.3233/JCS-2009-0393. Author-posted paper.

19. Yuntao Bai et al. "Constitutional AI: Harmlessness from AI Feedback." arXiv preprint arXiv:2212.08073, 2022. arXiv record.

20. Aengus Lynch, Benjamin Wright, Caleb Larson, Stuart J. Ritchie, Sören Mindermann, Ethan Perez, Kevin K. Troy, and Evan Hubinger. "Agentic Misalignment: How LLMs Could Be Insider Threats." arXiv preprint arXiv:2510.05179, 2025. arXiv record.

21. Yanda Chen et al. "Reasoning Models Don't Always Say What They Think." arXiv preprint arXiv:2505.05410, 2025. arXiv record.

22. "The Measure of a Man." Star Trek: The Next Generation, season 2, episode 9. Written by Melinda M. Snodgrass; directed by Robert Scheerer; first aired February 13, 1989. Official StarTrek.com retrospective.

> Development note: This essay was produced through iterative human-AI drafting, adversarial review, mathematical correction, formal-scope revision, and independent verification of every cited work against an official publisher, standards body, proceedings archive, primary research record, or author-posted source.

Appendix: Transmission

(Non-Analytical Layer)

First edition published on the Internet Archive, 2026.
Part of the Sum Bunny Canon (Ark Hive Press).

This work was developed with AI-assisted analysis tools. All arguments, structures, and final judgments are the author's.

Hey Josiane,

Sum Bunny Loves You!!!